Portable EEG

Darian Smith, Computer Engineering

Project Advisor: Dr. Anthony Richardson

April 1, 2012

Evansville, Indiana

Acknowledgements

I would first like to thank Dr. Anthony Richardson for assisting with finding the correct hardware for the project and the helpful hints for device communications. I would also like to thank Dr. Lora Becker for her assistance in explaining the ethical responsibilities when dealing with psychological data.

Table of Contents

List of Figures

List of Tables

I. Introduction

Throughout the day, many emotions are developed by the human mind. These emotions have been found to develop in the same locations of the brain for every person. Emotions and other thoughts are carried by small electrical charges along neural pathways. Devices that can retrieve the brain activity of someone and relay the data to the user via an easily understood interface are difficult to find. These electrical charges inside the brain can be measured by electroencephalogram (EEG) devices. An EEG is defined as a record of the tiny electrical impulses produced by the brain's activity. By measuring characteristic wave patterns, the EEG can help diagnose certain conditions of the brain. These devices are rarely used because of their inconvenience, expense, and lack of portability. To make an EEG device more practical the machine needs to be physically scaled down for the uses of diagnosing or assisting the user. By increasing the convenience, the interface of the EEG device is more user-friendly and geared toward functionality. After lowering the expense of the EEG, the device becomes household purchase and can be adapted toward long-term use. Once the portability of the EEG increases, the device can be taken to the user and be easily assessable instead of the potential user traveling to the device and having limited access.

II. Statement of the Problem

Nursing homes and senior care facilities do not have large incomes and do not have up-to-date technologies to care for their patients without needing to leave the facility. By scaling down the EEG device in both size and cost, facilities such as these can afford EEG equipment and easily transport the device to the resident. Assisting a resident is needed by nursing facilities for the individuals that are not able to speak or communicate with the staff of the building. Some

patients are also known to hide their emotions and seclude themselves from the rest of the community. To improve the residence's overall health, the nursing staff attempt to plan events and activities to get them active and maintaining their mental health through social interaction. To get the more resistant residents to take part in these activities, the staff must know the residence's preferences. If the individual consents to using the EEG device, the staff of the facility would be able to monitor the user and uncover what the resident enjoys.

Some consumers are simply curious how their own minds work. Other than going to the nearest hospital or purchasing a more than two thousand dollar item, people can only speculate on their neural activity. For others they want a new challenge that can take their cognitive ability to the next level.

Client requirements include the ease of use, comfort, portability, functionality, and recording of emotions. The device should send both signals of individual nodes and emotional levels to the computing unit. The computing unit is to receive the signal from the device and save it to memory. When the computing unit is connected to a visual display via HDMI, a user interface displays preprogrammed functions such as the instantaneous level of a chosen emotion and level of a chosen emotion over time. The case holding the computing unit is to protect the board from everyday use and function as a signal indicator.

After the user correctly places the neuroheadset onto the head, the portable user interface tells the user which nodes of the headset are retrieving a low amount of signals. The user must them adjust each low signal node to have great reception. When ready, the user starts the desired program and the neural data begins the process of being saved to memory. The data that is saved to memory is used to plot values of emotional intensity over time. This data can be used to see

emotional inconsistencies over long durations of time and pin point what events cause emotional discomfort.

III. Design Approach

The EEG is made cheaper by finding a low cost neural interface and computing unit. The EEG device is separated into two parts; the headset and the base. For the headset, the Emotiv® EEG neuroheadset was chosen for its convenience of Bluetooth®, price tag of three hundred dollars, and its mobile construction [1]. For the base, the main computing unit is the Galileo for its small size and price tag of sixty dollars [2]. For the visual interface, the PICASO microLCD, uLCD-32PTU, was chosen for its touchscreen display and built-in processor. The data gathered by the device is stored on a 32 GB SD card [3].

Figure 1: Emotiv® EEG Neuroheadset

The Emotiv® EEG neuroheadset, displayed in Figure 1, has 14 sensors for accurate EEG readings and 2 references for positioning the headset. Communications with the headset is achieved through Bluetooth® allowing the user to have a greater range of motion. The

Bluetooth® communications is from the headset to a USB dongle that is connected to the Galileo. The headset has a build in battery that lasts up to twelve hours on a single charge [1].

The Galileo, displayed in Figure 2, has a 400MHz Intel®® Quark System on Chip (SoC) X1000, a 32-bit Intel® Pentium-class. The board has a micro SD card slot to extend memory an extra 32GB to hold the program and emotional data. The Galileo is able to run Ubuntu 12.04 to preserve the functionality of the EmoEngine®. The Galileo is powered by a King Max, 2200 mAh, 7.4V battery. The battery is regulated down to 5V through a regulator and physically connected to the Vin and ground pins. A 3V coin cell battery is attached to the COIN pins on the Galileo board to run the real time clock whenever the device is powered down.

Figure 2: Galileo Development Board

The PICASO microLCD, displayed in Figure 3, is equipped with a touch-screen display to allow the user to interface solely with the screen. The user is able to choose a user profile, choose a program, control the chosen program, and watch the graphical interface display the

emotional data gathered. The micro LCD uses serial communication to receive the data from the Galileo. The display is powered from the +5 volt pin from the Galileo [3].

Figure 3: PICASO microLCD

The Emotiv® headset is worn on the head facing either forward or backward depending on if the user is actively in motion or resting. The computing unit is small enough to clip onto the belt or fit into a pocket. When the charge is low, the battery of the headset needs to be recharged and the batteries for the computing unit are to be changed. The computing unit must stay within 10 meters of the headset for the Bluetooth® technology to communicate properly. Both the headset and the computing unit should never be submerged in any liquid and should not operate near strong electromagnetic fields.

The emotions that are recorded by the program are instantaneous and long-term excitement and engagement. The level of excitement correlates to the user's activity in the

sympathetic nervous system. The physical response to this emotion includes pupil dilation, sweat glad stimulation, heart rate increases, and digestive inhibition. Related emotions to excitement are titillation, nervousness and agitation. The level of engagement correlates to the level of alertness and attention. Engagement is found through alpha and beta waves inside the brain. Related emotions to engagement are alertness, vigilance, concentration, stimulation, and interest [4].

The Emotiv® EEG interacts with computer, the Galileo saves or process those signals, and the PICASO microLCD presents the data on an interface. All the DLL files needed to run the program are installed into the computing unit allowing for the EmoEngine® to work. The EmoEngine® is part of the SDK that came with the Emotiv® EEG neuroheadset, allowing the signals that are sent by the headset to be converted to the magnitude of the emotions. A diagram of how the EmoEngine® works is presented in Figure 4 [4].

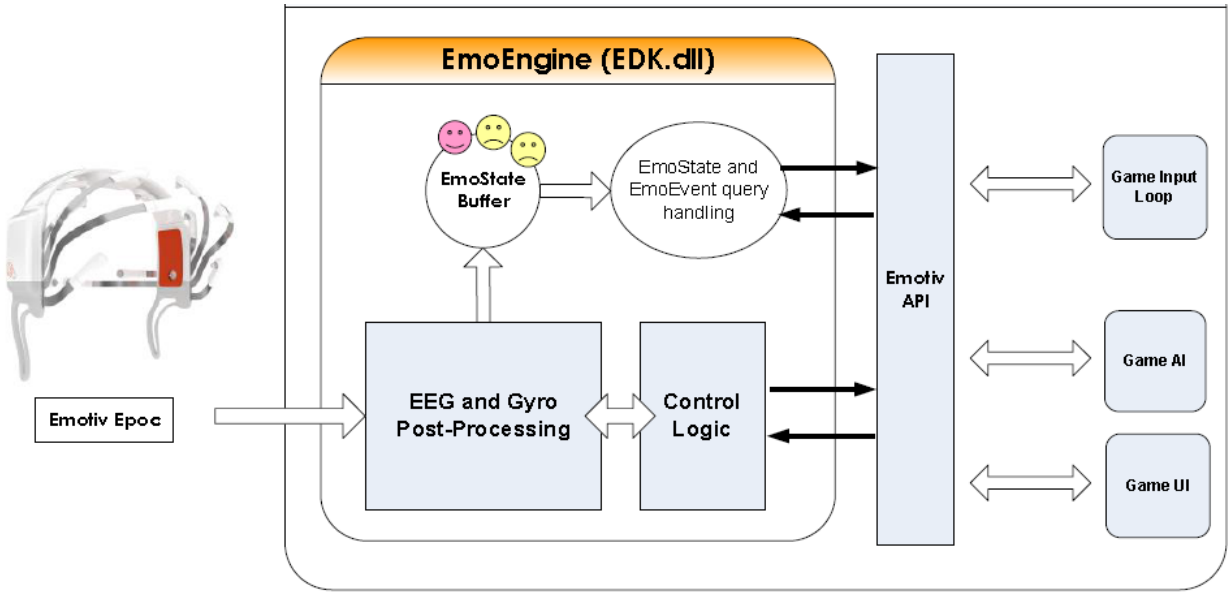

Figure 4: Use of the EmoEngine® [4]

The Linux operating is able to run on the Galileo enabling the running of the Emotiv® libraries. The functions that were supplied with research package that came with the EEG Emotiv® headset work most efficiently when the processor can supply 2.4 GHz [4]. The first program that is developed for the Emotiv® headset is an emotional logger which records emotions and places a time stamp to each recorded emotion. The emotional logger program is developed in C++ and can be viewed in appendix A.

The following is pseudo code for a portion of the emotional state logger program (Figure 5):

```
LOOP infinitely
WHILE a key is not hit on the keyboard
    Retrieve the next state from the Emotiv® headset
    IF the headset state is functional
        Retrieve the next event type and user ID
        IF the current event is updated from the previous event
            Retrieve emotional state from the user and the current time
            Save the emotional state with user ID to memory
        ENDIF
    ENDIF
    Do nothing until new data is available
ENDWHILE
```

Figure 5: Sample of Emotional State Logger Code

```
while (!_kbhit()) {

        state = EE_EngineGetNextEvent(eEvent);

        // New event needs to be handled
        if (state == EDK_OK) {

                EE_Event_t eventType = EE_EmoEngine®EventGetType(eEvent);
                EE_EmoEngine®EventGetUserId(eEvent, &userID);

                // Log the EmoState if it has been updated
                if (eventType == EE_EmoStateUpdated) {

                        EE_EmoEngine®EventGetEmoState(eEvent, eState);
                        const float timestamp = ES_GetTimeFromStart(eState);

                        logEmoState(ofs, userID, eState);        // ofstream ofs(argv[1])
                }
        }
        Sleep(1);
}
```

The code written to communicate from the Galileo development board to the PICASO microLCD is used to send the values associated with the intensity of each emotion. The values are send as hexadecimal digits where they are then displayed to the microLCD screen in the form of a line graph. This program was developed in C++ and interfaces with the Arduino libraries and can be viewed in appendix B. This program utilizes the functions that are constructed in

genieArduino.cpp and are described in the header file genieArduino.h that can be found in appendix C. This code parses the text file holding the emotional data, matches the baud rate of the PICASO microLCD, and sends the data serially.

The language developed by 4D Systems that enables their LCD screens to function is cross-bred between the C and Pascal programming languages. The code developed for the PICASO microLCD initializes with a keyboard interface and waits for the user to enter an ID. After the User enters an ID, the program begins to receive the data being sent serially from the Galileo development board and places the value into the current point of the line graph. The code for the user interface of the PICASO microLCD screen can be found in appendix D.

The client desires the device to be easy to move and use while being a reasonable price. The computing unit records either emotional or brain wave data and save in memory to view after the recording sessions has finished. The time stamp for every neural pattern recorded provide the time of day of which it had occurred and be sent with the saved data to the user-interface where the data can be easily read and understood by the consumer.

All wireless technology used in this project comply with Part 15 of the FCC safety requirements. The Emotiv® EEG neuroheadset cannot harm the user if worn in regular intervals of less than twelve hours. The removable nodes that touch the user's head is stored in a hydrator pack of saline solution to keep the equipment sterile. The Emotiv® EEG neuroheadset complies with the requirements of the Low Voltage Directive 2006/95/EC, the Electromagnetic Compatibility Directive 2004/108/EC, the Radio and Telecommunications Terminal Equipment Directive 1999/5/EC, and carries the CE and C-Tick marks [4].

IV. Results

The result is a product that consists of the Emotiv® EEG neuroheadset, the Galileo, and a user interface. The waveform given by the user interface reflects the emotional state of the user through the day. The headset communicates via Bluetooth® with the Galileo for the duration of the user's neural recording session and save all the data to external memory, i.e. SD card. The EEG package is hands-free to allow the user to participate in a normal, low-activity environment. The user is able to see the connectivity strength of each node from the headset and adjust each node for optimum neural interface readings. Each recording session starts and stop via the controls located on the PICASO LCD and is independent of the booting of the Galileo board. During testing, the system, as a whole, is able to last an hour under a single charge with the bottleneck being the battery attached to the Galileo development board. The PICASO microLCD and Galileo board pulled more current than originally expected. The Emotiv® neuroheadset is able to last 12 hours as expected. The battery attached to the Galileo board would need to have about ten times the amount of mAh to last as long as the headset. The Portable EEG is able to display the intensity of each emotion by reading the values created by the emotional state logger that is saved to a text file inside of the Galileo's SD card and the value is sent to the PICASO microLCD where the value is displayed onto a line graph over time.

The manufacturing of a single Portable EEG will consist of an Emotiv® neural EEG headset, Galileo development board, PICASO microLCD, 2 GB and 32 GB micro SD cards, 2200 mAh 7.2 V battery, and 3 V coin battery. All these item together cost $496.68 which is under the initial goal for the cost of a single unit. The cost of each part of the unit is given in Table 1.

V. Conclusion

To make an EEG device more practical the machine needs to be physically scaled down for the uses of diagnosing or assisting the user. By increasing the convenience, the interface of the EEG device is more user-friendly and geared toward functionality. After lowering the expense of the EEG, the device becomes household purchase and can be adapted toward long-term use.

Unit	Price
Emotiv Neuroheadset:	$299.00
Galileo Dev Board:	$59.79
3.2” PICASO microLCD:	$84.95
2 GB SD Card:	$6.35
32 GB SD Card:	$19.99
2200 mAh 7.4 V Battery:	$21.60
3 V Coin Battery:	Approx. $5.00
Total cost:	$496.68

Table 1: Cost of Single Portable EEG

Appendix A

This appendix includes the code used for the emotional state program.

```
#include <iostream>
#include <fstream>
#include <conio.h>
#include <sstream>
#include <windows.h>
#include <map>

#include "EmoStateDLL.h"
#include "edk.h"
#include "edkErrorCode.h"

#pragma comment(lib, "../lib/edk.lib")

void logEmoState(std::ostream& os, unsigned int userID, EmoStateHandle eState);


int main(int argc, char** argv) {

	EmoEngine®EventHandle eEvent		= EE_EmoEngine®EventCreate();
	EmoStateHandle eState			= EE_EmoStateCreate();
	unsigned int userID			= 0;
	const unsigned short composerPort	= 1726;
	int option = 0;
	int state  = 0;
	std::string input;

	try {

		if (argc != 2) {
			throw std::exception("Please supply the log file name.\nUsage: EmoStateLogger
			[log_file_name].");
		}

		std::cout << "Press '1' to start and connect to the EmoEngine®		" << std::endl;
		std::cout << "Press '2' to connect to the EmoComposer		" << std::endl;
		std::cout << ">> ";

		std::getline(std::cin, input, '\n');
		option = atoi(input.c_str());

		switch (option) {
			case 1:
			{
				if (EE_EngineConnect() != EDK_OK) {
					throw std::exception("Emotiv® Engine start up failed.");
				}
				break;
```

```
		}
		case 2:
		{
			std::cout << "Target IP of EmoComposer? [127.0.0.1] ";
			std::getline(std::cin, input, '\n');

			if (input.empty()) {
				input = std::string("127.0.0.1");
			}

			if (EE_EngineRemoteConnect(input.c_str(), composerPort) !=
			EDK_OK) {
				std::string errMsg = "Cannot connect to EmoComposer on [" +
				input + "]";
				throw std::exception(errMsg.c_str());
			}
			break;
		}
		default:
			throw std::exception("Invalid option...");
			break;
}


std::cout << "Start receiving EmoState! Press any key to stop logging...\n" << std::endl;

std::ofstream ofs(argv[1]);

while (!_kbhit()) {

		state = EE_EngineGetNextEvent(eEvent);

		// New event needs to be handled
		if (state == EDK_OK) {

			EE_Event_t eventType = EE_EmoEngine®EventGetType(eEvent);
			EE_EmoEngine®EventGetUserId(eEvent, &userID);

			// Log the EmoState if it has been updated
			if (eventType == EE_EmoStateUpdated) {

				EE_EmoEngine®EventGetEmoState(eEvent, eState);
				const float timestamp = ES_GetTimeFromStart(eState);

				printf("%10.3fs : New EmoState from user %d ...\r", timestamp,
				userID);

				logEmoState(ofs, userID, eState);
			}
		}
		else if (state != EDK_NO_EVENT) {
```

```
                        std::cout << "Internal error in Emotiv® Engine!" << std::endl;
                        break;
                }

                Sleep(1);
            }

            ofs.close();
        }
        catch (const std::exception& e) {
            std::cerr << e.what() << std::endl;
            std::cout << "Press any key to exit..." << std::endl;
            getchar();
        }

        EE_EngineDisconnect();
        EE_EmoStateFree(eState);
        EE_EmoEngine®EventFree(eEvent);

        return 0;
}


void logEmoState(std::ostream& os, unsigned int userID, EmoStateHandle eState) {

        // Log the time stamp and user ID
        os << ES_GetTimeFromStart(eState) << ",";
        os << userID << ",";
        os << static_cast<int>(ES_GetWirelessSignalStatus(eState)) << ",";

        // Expressiv Suite results
        os << ES_ExpressivIsBlink(eState) << ",";
        os << ES_ExpressivIsLeftWink(eState) << ",";
        os << ES_ExpressivIsRightWink(eState) << ",";

        os << ES_ExpressivIsLookingLeft(eState) << ",";
        os << ES_ExpressivIsLookingRight(eState) << ",";

        std::map<EE_ExpressivAlgo_t, float> expressivStates;

        EE_ExpressivAlgo_t upperFaceAction = ES_ExpressivGetUpperFaceAction(eState);
        float              upperFacePower  = ES_ExpressivGetUpperFaceActionPower(eState);

        EE_ExpressivAlgo_t lowerFaceAction = ES_ExpressivGetLowerFaceAction(eState);
        float              lowerFacePower  = ES_ExpressivGetLowerFaceActionPower(eState);

        expressivStates[ upperFaceAction ] = upperFacePower;
        expressivStates[ lowerFaceAction ] = lowerFacePower;

        os << expressivStates[ EXP_EYEBROW     ] << ","; // eyebrow
        os << expressivStates[ EXP_FURROW      ] << ","; // furrow
```

```
        os << expressivStates[ EXP_SMILE       ] << ","; // smile
        os << expressivStates[ EXP_CLENCH      ] << ","; // clench
        os << expressivStates[ EXP_SMIRK_LEFT  ] << ","; // smirk left
        os << expressivStates[ EXP_SMIRK_RIGHT ] << ","; // smirk right
        os << expressivStates[ EXP_LAUGH       ] << ","; // laugh

        // Affectiv Suite results
        os << ES_AffectivGetExcitementShortTermScore(eState) << ",";
        os << ES_AffectivGetExcitementLongTermScore(eState) << ",";

        os << ES_AffectivGetEngagementBoredomScore(eState) << ",";

        // Cognitiv Suite results
        os << static_cast<int>(ES_CognitivGetCurrentAction(eState)) << ",";
        os << ES_CognitivGetCurrentActionPower(eState);

        os << std::endl;
}
```

Appendix B

This appendix includes the Arduino code used inside the Galileo development board which parses and displays the text file that is created by the emotional state logger program.

```
//Developed by Darian Smith
#include <SD.h>
#include <genieArduino.h>

 File myFile;

void setup()
{
  genieBegin (GENIE_SERIAL_1, 9600);  //Serial1

   //Reset the Display
  //digitalWrites must be reversed as Display Reset is Active Low, and
  //the 4D Arduino Adaptors invert this signal so must be Active High.
  pinMode(4, OUTPUT);  // Set D4 on Arduino to Output
  digitalWrite(4, 1);  // Reset the Display via D4
  delay(100);
  digitalWrite(4, 0);  // unReset the Display via D4

  delay (3500); //let the display start up

  //Turn the Display on
  genieWriteContrast(1); // 1 = Display ON, 0 = Display OFF

   genieWriteStr(0, "Initializing SD card...");
    pinMode(10, OUTPUT);

   if (!SD.begin(4)) {
     genieWriteStr(0, "initialization failed!");
     return;
   }
   genieWriteStr(0, "initialization done.");

   // open the file for reading:
   myFile = SD.open("Emotiv®TextLog.txt", FILE_READ);
   if (myFile) {
     genieWriteStr(0, "Emotiv®TextLog.txt:");
     int emoValue[80][3];
     char checker[20];
     //get rid of first line of file
     while(!isdigit(myFile.peek())){
       myFile.read();
     }
     // read from the file until there's nothing else in it:
     int digitCursor=0, fileSection;
     char tester[1];
     while (myFile.available()) {
```

```
      do{
        while (myFile.peek()!=',' && myFile.peek()!='\n'){
          if (fileSection>=15 && fileSection<18){
            if(myFile.read()=='0'){
              if(myFile.peek()=='.'){
                myFile.read();  //take out '.'
                int theVal = 0;
                theVal = theVal + 10*((myFile.read())-'0');
                theVal = theVal + ((myFile.read())-'0');
                genieWriteObject(GENIE_OBJ_SCOPE, 0X00, theVal);
                while(myFile.peek()!=',' && myFile.peek()!='\n'){
                  myFile.read();
                }
              }
            }else{
              while(myFile.peek()!=',' && myFile.peek()!='\n'){
                myFile.read();
              }
              genieWriteObject(GENIE_OBJ_SCOPE, 0X00, 0);
            }
          }else{
            myFile.read();
          }
        }
        fileSection++;
        fileSection=fileSection%20;
        myFile.read();
      }while(fileSection!=0);
     digitCursor++;
     myFile.read(); //get rid of '\n'
   }

   myFile.close();
} else {
   // if the file didn't open, print an error:
   genieWriteStr(0, "error opening EmotivTestLog.txt");
 }
}

void loop()
{
    // nothing happens after setup
}
```

Appendix C

This appendix includes the header file that is implemented by the Arduino program that send the data from the Galileo development board to the PICASO microLCD.

```
////////////////////// GenieArduino 31/01/2014 //////////////////////
//
//    Library to utilize the 4D Systems Genie interface to displays
//    that have been created using the Visi-Genie creator platform.
//    This is intended to be used with the Arduino platform.
//
//                Improvements/Updates by
//                Clinton Keith, January 2014, www.clintonkeith.com
//                4D Systems Engineering, January 2014, www.4dsystems.com.au
//                4D Systems Engineering, September 2013, www.4dsystems.com.au
//                Written by
//                Rob Gray (GRAYnomad), June 2013, www.robgray.com
//    Based on code by
//                Gordon Henderson, February 2013, <projects@drogon.net>
//
//    Copyright (c) 2012-2013 4D Systems Pty Ltd, Sydney, Australia
/*********************************************************************
 * This file is part of genieArduino:
 *   genieArduino is free software: you can redistribute it and/or modify
 *   it under the terms of the GNU Lesser General Public License as
 *   published by the Free Software Foundation, either version 3 of the
 *   License, or (at your option) any later version.
 *
 *   genieArduino is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU Lesser General Public License for more details.
 *
 *   You should have received a copy of the GNU Lesser General Public
 *   License along with genieArduino.
 *   If not, see <http://www.gnu.org/licenses/>.
 *********************************************************************/
#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

#include <inttypes.h>

#include <stdint.h>

#ifndef genieArduino_h
#define genieArduino_h

#undef  GENIE_DEBUG
```

```
#define GENIE_VERSION       "GenieArduino 31-Jan-2014"

// Genie commands & replys:

#define GENIE_ACK           0x06
#define GENIE_NAK           0x15

#define TIMEOUT_PERIOD      1000
#define RESYNC_PERIOD       100

#define GENIE_READ_OBJ              0
#define GENIE_WRITE_OBJ             1
#define GENIE_WRITE_STR             2
#define GENIE_WRITE_STRU            3
#define GENIE_WRITE_CONTRAST        4
#define GENIE_REPORT_OBJ            5
#define GENIE_REPORT_EVENT                  7

// Objects
//      the manual says:
//              Note: Object IDs may change with future releases; it is not
//              advisable to code their values as constants.

#define GENIE_OBJ_DIPSW             0
#define GENIE_OBJ_KNOB              1
#define GENIE_OBJ_ROCKERSW                  2
#define GENIE_OBJ_ROTARYSW                  3
#define GENIE_OBJ_SLIDER            4
#define GENIE_OBJ_TRACKBAR                  5
#define GENIE_OBJ_WINBUTTON         6
#define GENIE_OBJ_ANGULAR_METER             7
#define GENIE_OBJ_COOL_GAUGE        8
#define GENIE_OBJ_CUSTOM_DIGITS             9
#define GENIE_OBJ_FORM              10
#define GENIE_OBJ_GAUGE             11
#define GENIE_OBJ_IMAGE             12
#define GENIE_OBJ_KEYBOARD                  13
#define GENIE_OBJ_LED               14
#define GENIE_OBJ_LED_DIGITS                15
#define GENIE_OBJ_METER             16
#define GENIE_OBJ_STRINGS                   17
#define GENIE_OBJ_THERMOMETER               18
#define GENIE_OBJ_USER_LED                  19
#define GENIE_OBJ_VIDEO             20
#define GENIE_OBJ_STATIC_TEXT       21
#define GENIE_OBJ_SOUND             22
#define GENIE_OBJ_TIMER             23
#define GENIE_OBJ_SPECTRUM                  24
#define GENIE_OBJ_SCOPE             25
#define GENIE_OBJ_TANK              26
```

```
#define GENIE_OBJ_USERIMAGES    27
#define GENIE_OBJ_PINOUTPUT             28
#define GENIE_OBJ_PININPUT              29
#define GENIE_OBJ_4DBUTTON              30
#define GENIE_OBJ_ANIBUTTON             31
#define GENIE_OBJ_COLORPICKER 32
#define GENIE_OBJ_USERBUTTON    33

// Structure to store replys returned from a display

#define         GENIE_FRAME_SIZE 6

struct genieFrameReportObj {
        uint8_t         cmd;
        uint8_t         object;
        uint8_t         index;
        uint8_t         data_msb;
        uint8_t         data_lsb;
};

//////////////////////////////////////////////////////////////////
// The Genie frame definition
//
// The union allows the data to be referenced as an array of uint8_t
// or a structure of type genieFrameReportObj, eg
//
//      genieFrame f;
//      f.bytes[4];
//      f.reportObject.data_lsb
//
//      both methods get the same byte
//
union genieFrame {
        uint8_t                         bytes[GENIE_FRAME_SIZE];
        genieFrameReportObj     reportObject;
};

#define MAX_GENIE_EVENTS        16      // MUST be a power of 2
#define MAX_GENIE_FATALS        10

struct genieEventQueueStruct {
        genieFrame      frames[MAX_GENIE_EVENTS];
        uint8_t         rd_index;
        uint8_t         wr_index;
        uint8_t         n_events;
};

typedef enum {
   GENIE_NULL,
   GENIE_SERIAL,
   GENIE_SERIAL_1,
```

```
   GENIE_SERIAL_2,
   GENIE_SERIAL_3
} genie_port_types;


//typedef void          (*geniePutCharFuncPtr)          (uint8_t c, uint32_t baud);
//typedef uint16_t      (*genieGetCharFuncPtr)                  (void);
typedef void            (*genieUserEventHandlerPtr) (void);

////////////////////////////////////////////////////////////////
// User API functions
// These function prototypes are the user API to the library
//
extern void             genieSetup              (uint32_t baud);
extern void             genieBegin              (Stream &serial);
extern uint16_t  genieBegin             (uint8_t port, uint32_t baud);
extern bool             genieReadObject         (uint16_t object, uint16_t index);
extern uint16_t         genieWriteObject        (uint16_t object, uint16_t index, uint16_t data);
extern void             genieWriteContrast      (uint16_t value);
extern uint16_t         genieWriteStr           (uint16_t index, char *string);
extern uint16_t         genieWriteStrU          (uint16_t index, uint16_t *string);
extern bool             genieEventIs            (genieFrame * e, uint8_t cmd, uint8_t object, uint8_t
index);
extern uint16_t  genieGetEventData      (genieFrame * e);
extern uint16_t         genieDoEvents           (void);
extern void             genieAttachEventHandler (genieUserEventHandlerPtr userHandler);
extern bool             genieDequeueEvent       (genieFrame * buff);
extern void             pulse                   (int pin);
extern void             assignDebugPort         (Stream &port);

#ifndef  TRUE
#define TRUE   (1==1)
#define FALSE (!TRUE)
#endif

#define ERROR_NONE                      0
#define ERROR_TIMEOUT           -1      // 255  0xFF
#define ERROR_NOHANDLER                 -2       // 254  0xFE
#define ERROR_NOCHAR            -3      // 253  0xFD
#define ERROR_NAK               -4      // 252  0xFC
#define ERROR_REPLY_OVR         -5      // 251  0xFB
#define ERROR_RESYNC            -6      // 250  0xFA
#define ERROR_NODISPLAY         -7      // 249  0xF9
#define ERROR_BAD_CS            -8      // 248  0xF8

#define GENIE_LINK_IDLE         0
#define GENIE_LINK_WFAN         1 // waiting for Ack or Nak
#define GENIE_LINK_WF_RXREPORT      2 // waiting for a report frame
#define GENIE_LINK_RXREPORT             3 // receiving a report frame
#define GENIE_LINK_RXEVENT              4 // receiving an event frame
#define GENIE_LINK_SHDN         5
```

```
#define GENIE_EVENT_NONE        0
#define GENIE_EVENT_RXCHAR      1

#endif
```

Appendix D

This appendix includes the code for the keyboard interface of the user and transitions to the graphical interface presenting a line graph of the value of intensity of each emotion over time.

```
#platform "uLCD-32PTU"
// Program Developed by Darian Smith
// Program Skeleton 1.1 generated 4/12/2014 8:05:44 PM

#inherit "4DGL_16bitColours.fnc"
#inherit "VisualConst.inc"
#inherit "KeyboardConst.inc"
#inherit "Scope4.inc"
#inherit "KBRoutines.inc"
#inherit "FANCYBUTTONSConst.inc"

var old_y1[220], new_y1[220] ;
var old_y2[220], new_y2[220] ;
var old_y3[220], new_y3[220] ;
var old_y4[220], new_y4[220] ;

var i, k, j, x, KeyPointer, y, KeyString[30], cursor, printPosition, exit, saveKey, state;

func KbHandler(var Key)
   txt_MoveCursor(0,7);
   if(Key == 8)   //backspace
      if(cursor > 0)
         cursor--;
      endif
      KeyString[cursor] :=  ' ';
      for(printPosition:=0; printPosition<=cursor; printPosition++)
        print([CHR]KeyString[printPosition]);
      next
   else if (Key == 13) // 'enter'
      saveKey := Key;
      exit := 1;
   else
      txt_MoveCursor(0,7);
      KeyString[cursor] := Key;
      for(printPosition:=0; printPosition<=cursor; printPosition++)
        print([CHR]KeyString[printPosition]);
      next
      //print([CHR] Key);
      cursor++;
   endif
endfunc

func main()
   var i, state, n ;

   putstr("Mounting...\n");          //SD mounting
```

```
if (!(disk:=file_Mount()))
   while(!(disk :=file_Mount()))
      putstr("Drive not mounted...");
      pause(200);
      gfx_Cls();
      pause(200);
   wend
endif
gfx_TransparentColour(0x0020);
gfx_Transparency(ON);

hndl := file_LoadImageControl("STRING~1.dat", "STRING~1.gci", 1);
//hndl := file_LoaderImageControl("FANCYB~1.dat", "", );

gfx_Set(SCREEN_MODE,LANDSCAPE) ;
img_Show(hndl,ikeyboard1) ; // show initialy, if required
for (i := ikeyboard1+1; i <= ikeyboard1+okeyboard1[KbButtons]; i++)
      img_SetWord(hndl, i, IMAGE_FLAGS, (img_GetWord(hndl, i, IMAGE_FLAGS) |
      I_STAYONTOP) & ~I_TOUCH_DISABLE); // set to enable touch, only need to do this once
next

touch_Set(TOUCH_ENABLE);                    // enable the touch screen

txt_MoveCursor(2,7);
print("Press 'Enter' when ready.");

repeat

   state := touch_Get(TOUCH_STATUS);          // get touchscreen status
   n := img_Touched(hndl,-1) ;

   //-------------------------------------------------------------------------------------

   if(state == TOUCH_PRESSED)                 // if there's a press
      x := touch_Get(TOUCH_GETX);
      y := touch_Get(TOUCH_GETY);
      if ((n >= ikeyboard1) && (n <= ikeyboard1+okeyboard1[KbButtons]))
         kbDown(ikeyboard1, okeyboard1, ikeyboard1keystrokes, n-ikeyboard1, KbHandler) ;
      endif
   endif

   //-------------------------------------------------------------------------------------
   if(state == TOUCH_RELEASED)                // if there's a release
      if (okeyboard1[KbDown] != -1)
         kbUp(ikeyboard1, okeyboard1) ;
         procKey();
      endif
   endif

   //-------------------------------------------------------------------------------------
   if(state == TOUCH_MOVING)                  // if it's moving
```

```
        x := touch_Get(TOUCH_GETX);
        y := touch_Get(TOUCH_GETY);
      endif
   forever
endfunc

func procKey()
   if(saveKey == 13)
      // Form2 1.1 generated 4/13/2014 3:22:36 PM

      // prevButton 1.0 generated 4/13/2014 3:22:36 PM
      img_ClearAttributes(hndl, iprevButton, I_TOUCH_DISABLE); // set to enable touch
      img_Show(hndl, iprevButton);  // show button, only do this once
      img_SetWord(hndl, iprevButton, IMAGE_INDEX, state); // where state is 0 for up and 1 for down
      img_Show(hndl,iprevButton) ;

      // nextButton 1.0 generated 4/13/2014 3:22:36 PM
      img_ClearAttributes(hndl, inextButton, I_TOUCH_DISABLE); // set to enable touch
      img_Show(hndl, inextButton);  // show button, only do this once
      img_SetWord(hndl, inextButton, IMAGE_INDEX, state); // where state is 0 for up and 1 for down
      img_Show(hndl,inextButton) ;

      //take out strings
      txt_MoveCursor(0,7);
      print("                          ");
      txt_MoveCursor(2,7);
      print("                       ");

      // Scope1 1.0 generated 4/13/2014 3:22:36 PM

      // Create empty initial scope
      gfx_RectangleFilled(0, 40, 320, 240, BLACK) ;
      gfx_Hline(40 + 100, 0, 320, YELLOW) ;
      Graticule(0, 40, 320, 240, 10,  10, 0x0280) ;

        // draw and update scope
      Graticule(0, 40, 320, 240, 10,  10, 0x0280) ;
      gfx_Scope(0, 320, 240, 320, 0, 100, BLACK,
          old_y1, new_y1, LIME, old_y2, new_y2, BLUE, old_y3, new_y3, RED, old_y4, new_y4,
FUCHSIA  ) ;
      gfx_Hline(40 + 100, 0, 320, YELLOW) ;

      mem_Copy(&new_y1[k], new_y1, 440 - k*2);    // this will only work in R35 and above PmmC
      mem_Copy(&new_y2[k], new_y2, 440 - k*2);
      mem_Copy(&new_y3[k], new_y3, 440 - k*2);
      mem_Copy(&new_y4[k], new_y4, 440 - k*2);
      forever
   endif
endfunc
```

References

[1] Emotiv®, "EEG Features," Emotiv®, [Online]. Available: http://Emotiv®.com/eeg/features.php. [Accessed 8 December 2013].

[2] "Intel® Galileo1 Development Board," Newegg, [Online]. Available: http://www.newegg.com/Product/Product.aspx?Item=N82E16813121792. [Accessed 8 2 2014].

[3] "uLCD-32PTU," 4D Systems, [Online]. Available: http://www.4dsystems.com.au/product/1/9/4D_Intel®ligent_Display_Modules/uLCD_32PTU/. [Accessed 8 12 2013].

[4] Emotiv®, *Emotiv® Software Development Kit: User Manual for Release 2.0.0.20*, Hong Kong, San Francisco, Eveleigh: Emotiv®, 2013.

[5] "Galileo Getting Started Guide," Sparkfun, 2014. [Online]. Available: https://learn.sparkfun.com/tutorials/galileo-getting-started-guide. [Accessed 21 3 2014].